PredictRoute: A Network Path Prediction Toolkit

RACHEE SINGH, University of Massachusetts, Amherst, USA
DAVID TENCH, University of Massachusetts, Amherst, USA
PHILLIPA GILL, University of Massachusetts, Amherst, USA
ANDREW MCGREGOR, University of Massachusetts, Amherst, USA

Accurate prediction of network paths between arbitrary hosts on the Internet is of vital importance for network operators, cloud providers, and academic researchers. We present PredictRoute, a system that predicts network paths between hosts on the Internet using historical knowledge of the data and control plane. In addition to feeding on freely available traceroutes and BGP routing tables, PredictRoute optimally explores network paths towards chosen BGP prefixes. PredictRoute's strategy for exploring network paths discovers 4X more autonomous system (AS) hops than other well-known strategies used in practice today. Using a corpus of traceroutes, PredictRoute trains probabilistic models of routing towards prefixes on the Internet to predict network paths and their likelihood. PredictRoute's AS-path predictions differ from the measured path by at most 1 hop, 75% of the time. We expose PredictRoute's path prediction capability via a REST API to facilitate its inclusion in other applications and studies. We additionally demonstrate the utility of PredictRoute in improving real-world applications for circumventing Internet censorship and preserving anonymity online.

CCS Concepts: • **Mathematics of computing** → *Paths and connectivity problems*; *Markov networks*; • **Networks** → *Routing protocols*; *Wide area networks*; *Public Internet*.

Additional Key Words and Phrases: network measurement; Internet path prediction

ACM Reference Format:
Rachee Singh, David Tench, Phillipa Gill, and Andrew McGregor. 2021. PredictRoute: A Network Path Prediction Toolkit . *Proc. ACM Meas. Anal. Comput. Syst.* 5, 2, Article 23 (June 2021), 24 pages. https://doi.org/10.1145/3460090

1 INTRODUCTION

Despite its engineered nature, the Internet has evolved into a collection of networks with different–and sometimes conflicting–goals, where understanding its behavior requires empirical study of topology and network paths. This problem is compounded by networks' desire to keep their routing policies and behaviors opaque to outsiders for commercial or security-related reasons. Researchers have worked for over a decade designing tools and techniques for inferring network connectivity and paths [26, 27]. However, operators seeking to leverage information about network paths or researchers in need of Internet paths to evaluate new Internet-scale systems are often confronted with limited vantage points for direct measurement and myriad data sets, each offering a different lens on Internet connectivity and paths.

Predicting network paths is crucial for a variety of problems impacting researchers, network operators, content and cloud providers. Researchers often need knowledge of Internet routing to

Authors' addresses: Rachee Singh, University of Massachusetts, Amherst, USA; David Tench, University of Massachusetts, Amherst, USA; Phillipa Gill, University of Massachusetts, Amherst, USA; Andrew McGregor, University of Massachusetts, Amherst, USA.

2476-1249/2021/6-ART23 $15.00
https://doi.org/10.1145/3460090

evaluate Internet-scale systems *e.g.,* refraction routing [36], Tor [11], Secure-BGP [17]. For network operators, network paths can aid in diagnosing the root cause of performance problems like high latency or packet loss. Content providers debugging poor client-side performance require the knowledge of the set of networks participating in delivering client traffic to root-cause bottleneck links. While large cloud providers, like Amazon, Google and Microsoft are known to develop in-house telemetry for global network diagnostics, small companies, ISPs and academics often lack such visibility and data.

Understanding and predicting Internet routes is confounded by several factors. Internet paths are dependent on several deterministic but not public phenomena: route advertisements made via BGP and best path selection algorithms based on private business relationships. Additionally, factors like load balancing via ECMP, intermittent congestion on network links, control plane mis-configurations and BGP hijacks also impact network paths.

Standard diagnostic tools like traceroute provide limited visibility into network paths since the user can only control the destination of a traceroute query, the source being their own network. Tools like reverse traceroute [23] rely on the support of IP options to shed light on the reverse path towards one's network. In addition to requiring the support for IP options from Internet routers, these techniques require active probing from the client or a set of vantage points distributed on the Internet. Active probing is not only expensive in terms of amount of diagnostic traffic generated (traceroutes, pings *etc.*) but also provides limited visibility into the temporal dynamics of the network state *e.g.,* network path yesterday vs. the measured path today.

In this work, we design and develop PredictRoute, a system that predicts network paths between arbitrary sources and destinations on the Internet by developing probabilistic models of routing from observed network paths. For this purpose, PredictRoute, leverages existing data plane measurements, optimizes the use of data plane measurement platforms, and simulates routing behavior when empirical data is absent. In the median case, the path predicted by PredictRoute differs from the measured path by only 1 hop. We offer the code and a proof-of-concept implementation of PredictRoute at http://predictroute.github.io/. In its present form, the PredictRoute REST API allows users to query network paths between sources and destinations (BGP routed prefixes or autonomous systems). In addition to providing the predicted paths, PredictRoute provides *confidence* values associated with each network path based on historical information.

PredictRoute complements the approach of existing path-prediction systems [24, 26] by developing efficient algorithms for measuring the routing behavior towards all BGP prefixes on the Internet. When measuring paths towards each BGP prefix, PredictRoute maximizes discovery of the network topology with a constrained measurement budget, both globally and per vantage point. PredictRoute's strategy for exploring network paths discovers 4X more AS-hops than other well known strategies used in practice today.

1.1 Key Contributions

The problem of predicting network paths has been well studied, with important work like iPlane [26], iPlane Nano [27], BGPSim [18] and Sibyl [24] tackling different parts of the problem. Our key contributions in the context of these existing systems are the following:

Per-destination probabilistic Markov models. Systems like iPlane [26] and iPlane Nano [27] consume traceroutes to build an atlas of network paths. By combining *splices* of paths from this atlas, the systems predict a previously un-measured path. Recently, the prediction accuracy of iPlane was found to be low – 68% at the AS path level by Sibyl [24]. Sibyl proposes to improve the low accuracy of splicing-based network path prediction using supervised learning to choose between multiple possible paths.

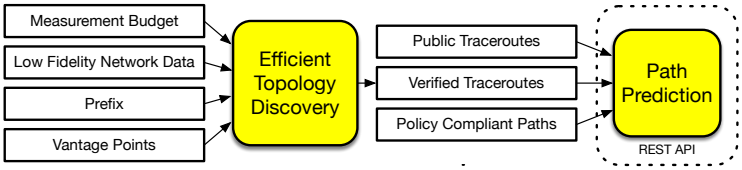


Fig. 1. PredictRoute achieves two goals: efficient topology discovery and accurate path prediction.

PredictRoute differs from the existing approaches by using the key fact that routing on the Internet is largely destination based [4]. This means, traceroutes through a network are likely to traverse the same paths towards a given destination prefix, irrespective of where they originated. Therefore, unlike previous path prediction systems, PredictRoute constructs a *destination-specific* probabilistic model for each destination prefix or autonomous system (AS) in place of a common atlas for all destinations. Using observations of network paths over time, PredictRoute not only infers the connectivity between networks but also learns the likelihood of picking different next-hops from a given network as a function of time.

Leveraging publicly available datasets. PredictRoute bootstraps the construction of destination-specific routing models using public traceroutes from RIPE Atlas [30] and CAIDA Ark [7]. While public traceroutes can be sparse, PredictRoute reuses existing crowd-sourced measurements from different platforms to reduce redundant diagnostic traffic from on-demand measurements. The availability of more public datasets for training will help PredictRoute achieve higher topological coverage of the Internet and answer more path prediction queries.

Efficient global and per-destination topology discovery. A related system, Sibyl [24], allows efficient use of measurement budget for answering path queries between sources and destination IP addresses. Instead, PredictRoute focusses on a special case of this problem: efficiently answering *all* queries towards a destination prefix for a fixed measurement budget. PredictRoute's approach for doing measurement selection is within constant factor of optimal in the general case. However, when the routing towards a prefix is destination based, with no violations, we show that PredictRoute's measurement selection is optimal.

Deployment of PredictRoute. We are releasing the entire codebase of PredictRoute and have deployed a proof-of-concept implementation of PredictRoute at http://predictroute.github.io/. The system is hosted on Azure's cloud platform for demonstration purposes. We use a modestly provisioned cloud instance (2 CPU cores, 4 GB RAM). The REST API can be incorporated programmatically into other systems.

2 THE PREDICTROUTE SYSTEM

Figure 1 shows the two major components of PredictRoute along with the input for each. These components are:

(1) *Efficient Topology Discovery.* PredictRoute makes efficient use of limited measurement resources to discover the network topology towards an IP prefix. PredictRoute's topology discovery algorithm takes as input a destination BGP prefix P, a set of vantage points V' capable of sending traceroutes to P, a measurement budget k, and information of P's network topology derived from BGP routing tables or stale traceroutes (§3.1). This topology discovery algorithm outputs a budget-compliant set of vantage points $S \subset V', |S| \leq k$ from which to measure to reveal as much information about paths towards P as possible (§3.2). In §3, we describe this algorithm and its strong performance guarantees in detail.

(2) *Path Prediction.* PredictRoute combines public traceroutes from measurement platforms like RIPE Atlas [30] and those run by its own topology discovery module to learn Markov models of routing towards each BGP routed prefix (§4). PredictRoute infers network paths between source s and destination prefix P from the Markov model (§4). While PredictRoute aims to explore as much of the network topology as it can via active measurements, it will still lack a global view due to absence of vantage points in different parts of the Internet. As a result, when a queried path (source s and destination P) cannot be inferred from P's empirical Markov model, PredictRoute will fall back to an algorithmic simulation of policy-compliant BGP paths [18]. We micro-benchmark different components of PredictRoute (§5) and extensively evaluate the accuracy and coverage of PredictRoute in comparison with previous work (§6).

2.1 Design choices

Granularity of network destinations. In previous work, systems for path prediction have predicted paths towards destinations of various granularities. BGPSim [18] gives BGP policy compliant paths towards *destination ASes*, whereas iPlane [26], iNano [27], and Sibyl [24] predict paths towards *destination hosts or prefixes*. PredictRoute predicts paths towards both AS-level and prefix-level destinations. PredictRoute's routing models for destinations are general and can be easily adapted to other destination granularities, *e.g.,* BGP atoms [3] and PoPs. For the applications that motivate PredictRoute, destinations at the granularity of ASes and prefixes are sufficiently informative.

Granularity of network paths. So far, we have not discussed what constitutes a network path predicted by PredictRoute. Existing path prediction approaches predict paths where each hop in the path is either an AS or a PoP. PredictRoute can predict paths at AS-level, BGP routed prefix-level or even at a /24 prefix granularity. The granularity of the hops in PredictRoute's routing models are configurable by the users of the system. The granularity of the predicted path can impact its utility for different applications. For instance, AS-level paths can be sufficiently informative for quantifying the threat of eavesdropping ISPs on anonymous communication [28, 33].

However, some ASes can be very large, spanning entire countries *e.g.,* Tier-1 networks like AT&T and Level3, making AS-level path prediction too coarse for diagnostic purposes. Inferring PoPs from router IP addresses is a research problem in its own right PredictRoute avoids introducing the complexity of PoP inference by predicting prefix-level paths. Thus, in the default case, PredictRoute predicts BGP-routed prefix-level paths. Prefix-level paths have sufficient information to predict AS-level paths by mapping prefixes back to ASes that announce them in BGP to obtain an AS-level path, if desired. A very *fine granularity* of hops in network paths can cause an explosion in the size of routing models built by PredictRoute and this problem gets worse when considering IPv6 paths due to the large IPv6 address space. In the later sections of this paper, we evaluate PredictRoute using IPv4 AS-level and BGP-routed prefix-level paths only.

Markov models for path prediction. Since paths on the Internet are an outcome of several unobservable and uncertain phenomenon, including, congestion, route failures, load balancing, time-dependent routing policies *etc.*, PredictRoute implements a *probabilistic model* for path prediction based on an empirical understanding of network paths. We build first order Markov chains, one for each routed prefix. While, higher order Markov chains can also be leveraged in PredictRoute, we make the first order Markov assumption since network paths on the Internet are short [10] and incorporating the current and previous hops in making routing decisions already accounts for a large part of the overall path. Second, computing transition probabilities with higher order Markov chains needs more data and relevant network measurements are scarce. Thus, we argue that first order Markov chains are a reasonable choice for PredictRoute's per-prefix routing models.

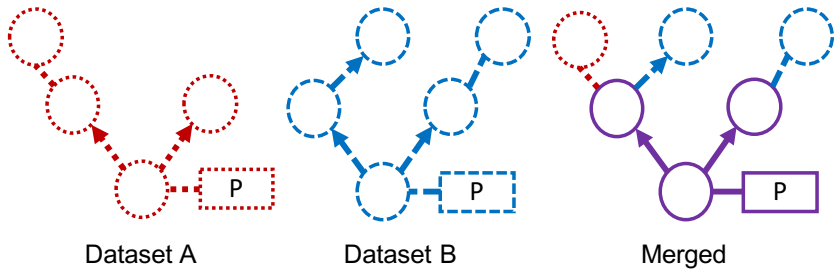


Fig. 2. DAGs for a destination prefix P in different traceroute datasets are merged to create a single DAG for P. Each circle represents an autonomous system.

With the routed prefix (or AS) as the *end state*, and prefixes (or ASes) as potential *start states* in the Markov chain, the problem of network path prediction is to find the highest probability sequence of states from a given start to the end state.

3 EFFICIENT TOPOLOGY DISCOVERY

In this section, we discuss the problem of discovering the AS-level topology towards a prefix. To *bootstrap* the topology discovery, we use publicly available traceroutes and simulated BGP policy-compliant paths from BGPSim (§3.1). While public traceroutes can be stale and BGP policy compliant paths may not be of high accuracy, they help PredictRoute develop an initial notion of the network topology to different destinations. We bootstrap the topology discovery problem from these two data sources since they are essentially *free* – do not require issuing active measurements from vantage points. We then frame the challenge of maximizing per-prefix topology discovery for a given measurement budget as an optimization problem, which we show is equivalent to a special case of the Max-Cover$_k$ (§3.2) problem when routing is destination-based. We present a greedy algorithm (§3.3) that optimally solves this topology discovery problem. In the case where ASes violate destination-based routing [13], we can still guarantee a constant-factor approximation of the problem using a relaxation of the Max-Cover$_k$ problem (§3.5).

3.1 Bootstrapping topology discovery

PredictRoute uses publicly available traceroutes on the RIPE Atlas [30] and CAIDA Ark [7] measurement platforms to bootstrap network topology discovery. Since public traceroutes can be intermittent and sparse, we gather them over a period of 10 days to gain a significant number of measurements for bootstrapping. The gathered data from RIPE Atlas's consists of traceroutes to hundreds of destination prefixes and Ark's data set has traceroutes to more than 300K prefixes.

From traceroutes to AS-level paths. Converting traceroutes from IP to AS-level paths is fraught with challenges [9]. We take a best-effort approach to this conversion but note that improved IP to AS path mapping solutions would only help the accuracy of PredictRoute. This section describes the heuristics we use to avoid common pitfalls in mapping traceroutes to AS-level paths. We use CAIDA's IP to AS dataset [1] to map each IP in the traceroute to an AS number. We then use PeeringDB [2] to remove IXP hops (based on the prefix and AS number) from our traceroutes. In the case of unresponsive hops in traceroutes we do not infer AS-connectivity. We discard circuitous traceroutes. More details on the conversion of IP paths into AS-level paths is in Appendix A.

Augmenting with BGP paths. In addition to using public traceroutes, PredictRoute also gathers BGP policy-compliant paths (derived from BGPSim [19]) for bootstrapping.

Merging measurements to form DAGs. We merge the AS-level paths from different datasets toward the same destination prefix to form a directed acyclic graph (DAG) for each destination prefix. Figure 2 illustrates how two DAGs for a destination prefix are merged into a single DAG. We note that the DAG will not be a tree when network paths violate destination-based routing. We observe 2% of source-destination pairs violate destination-based routing. In Sections 3.4 and 3.5, we describe how we deal with these cases. The DAGs constructed from measurements define the initial structure of the Markov chain for each prefix.

The coverage of a vantage point is the set of inter-AS links or edges in a prefix's initial DAG that a traceroute from this vantage point to the prefix would reveal. Before measuring from it, we need a good *estimate* of the coverage the vantage point can provide. PredictRoute uses the estimated coverage to decide the utility of measuring from the vantage point. PredictRoute uses the prefix DAGs constructed in the bootstrapping phase to derive estimates of the coverage of vantage points. The actual coverage of vantage points can differ from our estimates. For the remainder of this section we give performance guarantees on the assumption that the predicted and actual coverage of vantage points are identical. We justify this assumption and show that the estimated coverage from simulated paths matches that of measured paths in Appendix B.

3.2 Maximizing topology discovery

Since routing on the Internet is largely destination-based [4], we can represent the paths we know toward a given prefix as a DAG, as described in the previous section. We define the destination-based DAG of a prefix P as $G = (V, E)$, where V consists of P and all ASes on the Internet. Edge $e \in E$ represents a connection between two ASes observed in measurements or simulations. We consider the prefix as the root of this graph as opposed to the AS that announced it to account for per-prefix routing policies [4]. $V' \subset V$ is the set of ASes that have vantage points.

In practice, for a given prefix P, we do not know G. By executing traceroutes from a subset S of V', we obtain a *partial observation* of G denoted by $\hat{G} = (\hat{V}, \hat{E})$. $\hat{V}$ is composed of AS hops observed in traceroutes from ASes in V' to P. Edges in $\hat{E}$ consist of AS edges inferred from the traceroutes. Let the coverage of a set of measurements from $S \subset V'$ be:

$$\text{Cov}(S) = |\hat{E}| \tag{1}$$

Figure 3 illustrates one such prefix-based graph. The blue hashed nodes indicate networks containing vantage points (*e.g.,* RIPE Atlas probes) and the purple striped nodes indicate nodes that have been discovered in the prefix's DAG. The discovered nodes consist of the nodes directly observed in traceroutes towards P and all single-homed customer ASes [20] of the directly observed nodes. Here $\hat{V}$ is the set of shaded nodes and $\hat{E}$ are the edges that connect them. White nodes represent nodes that we cannot discover via measurements towards P due to the lack of suitable vantage points. We consider the single-homed customers of empirically discovered nodes to be discovered since single-homed ASes connect to the Internet only through one AS link.

This leads us to the following problem definition for exploring the largest portion of the AS topology (G) with a measurement budget of k traceroutes:

Problem 1 (Prefix-Cover$_k$). *Find $S \subset V'$, where $|S| \leq k$, that maximizes* Cov(S).

The Prefix-Cover$_k$ problem is reducible to the Max-Cover$_k$ problem in which the input is a number k and a collection $\mathcal{A} = \{A_1, A_2, \ldots\}$ of sets and the goal is to select a subset $\mathcal{A}' \subset \mathcal{A}$ of k sets that maximizes $|\cup_{A \in \mathcal{A}'} A|$. The reduction is immediate since the measurement from each vantage point can be thought of as a set A_i, and our goal is to maximize the coverage of edges by choosing k of these sets. Max-Cover$_k$ is unfortunately NP-Hard. A well-known greedy algorithm

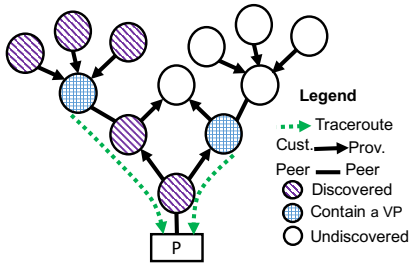


Fig. 3. An example of a prefix-based DAG.

Input: Graph $G = (V, E)$ with vantage points V'.
$S \leftarrow \emptyset$
for $i = 1$ *to* k **do**
  $S \leftarrow S + \text{argmax}_{s \in V'} \text{Cov}(S + s)$
end
return S

Algorithm 1: Greedy Vantage Point Selection

provides the best possible approximation with a factor of $(1 - 1/e) = 0.632\ldots$ of the optimal solution [12]. However, since Prefix-Cover$_k$ is a special case of Max-Cover$_k$, we find that it can be solved exactly which we discuss in the next section.

3.3 Optimality for Destination-based Routing

We now present a greedy algorithm to select vantage points to perform measurements from (Algorithm 1). We show that this algorithm finds the optimal solution to Prefix-Cover$_k$ when the graph G is a tree.

Theorem 1. *Algorithm 1 solves Prefix-Cover$_k$ exactly when G is a tree, i.e., when there are no violations of destination-based routing.*

Proof. Let $S = \{s_1, s_2, \ldots, s_k\}$ denote the subset of vantage points returned by the greedy algorithm on G, where s_1 denotes the first vantage point chosen, s_2, denotes the second, and so on. Recall that $\text{Cov}(S)$ denotes the coverage on G of the measurements run from S, and let $C(S)$ denote the corresponding set of edges that are discovered. For the sake of contradiction assume that for some $i < k$, there is an optimal solution T such that $s_1, \ldots, s_{i-1} \in T$ but no optimal solution contains $s_1, \ldots, s_i$. Let

$$T = \{s_1, s_2, \ldots, s_{i-1}, t_i, t_{i+1}, \ldots t_k\}$$

where $s_i \neq t_j$ for all $j \geq i$. In other words, we assume that for some i, the first $i - 1$ vantage points in S are also in T, but that s_i does not appear in T. For any set of vantage points A, define

$$C'(A) = C(A) \setminus C(s_1, s_2, \ldots, s_{i-1})$$

and $\text{Cov}'(A) = |C'(A)|$.

For any $j \geq i$, since s_i was chosen by the greedy algorithm before t_j we can infer that

$$\text{Cov}'(s_i) \geq \text{Cov}'(t_j)\ .$$

Let $j' \geq i$ be chosen to maximize $|C'(s_i) \cap C'(t_{j'})|$ and define $T' = (T \cup \{s_i\}) \setminus \{t_{j'}\}$. Observe that

$$\text{Cov}'(T') \geq \text{Cov}'(T) + \text{Cov}'(s_i) - \text{Cov}'(t_{j'})$$

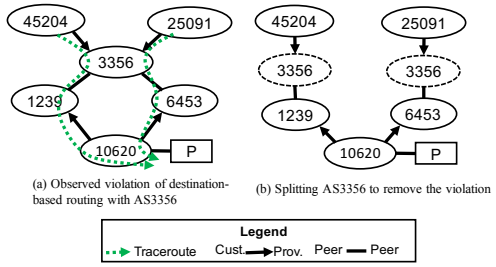


Fig. 4. Example of violation of destination-based routing. Depending on the prior hop AS 3356 (Level 3) selects a different next-hop towards the destination. Splitting this node produces a tree-structured prefix DAG.

Since $\text{Cov}'(s_i) \geq \text{Cov}'(t_{j'})$, we deduce that $\text{Cov}(T') \geq \text{Cov}(T)$ and so T' is also optimal. But $s_1, \ldots, s_i \in T'$ which is a contradiction to the assumption that no optimal solution contains $\{s_1, \ldots, s_i\}$. □

3.4 Extension for prior-hop violations of destination-based routing

When merging multiple traceroute-derived AS paths we observe cases that violate the assumption of destination-based routing. Figure 4 shows one such example. Here, we observe AS 3356 (Level 3) selecting different next-hop ASes towards the same BGP-routed prefix, depending on the prior hop in the path. To exclude the effect of churn in network paths, these traceroutes were run only a few minutes apart. In this case, it appears Level 3's routing decision is impacted by the prior AS hop, thus we can "split" the node into two nodes, each of which represents 3356's routing behavior for each of the prior hops. The resulting graph is a tree and Algorithm 1 can find the optimal measurement set to explore the graph. In general, we can split AS nodes that violate destination-based routing based on their prior AS hop by creating a copy of the node for each prior hop, and adding to each copied node all outgoing edges associated with that prior hop. The resulting tree has the same number of edges as the original graph.

3.5 Other violations of destination-based routing

Not all violations of destination-based routing are deterministically dependent only on the prior AS hop. In these cases, it is often difficult to determine exactly what rule underlies the routing behavior and we simply treat this behavior as a random process. Figure 5 shows an example where traceroutes passing through node 4 are randomly routed on the left link with probability 0.4 and routed on the right link otherwise. This behavior can arise when routers perform uneven load balancing over outgoing links. This gives rise to the problem of maximizing the *expected* coverage through our choice of vantage points. The set of edges covered by each vantage point v_i is now a random variable X_{v_i} whose value is the edges traversed by a random walk beginning at v_i and ending at P, where each step is chosen from the outgoing edges as per their routing probability. Thus, the problem of maximizing the expected coverage of a topology can be defined as:

Problem 2 (Stoc-Prefix-Cover$_k$). *Find $S \subset V'$ with $|S| \leq k$, that maximizes $\mathbb{E}\left[|\cup_{v \in S} X_v|\right]$.*

For example, in Figure 5, X_1, the expected coverage of vantage points in node 1, is determined by starting at node 1, continuing to nodes 3 and 4, and randomly choosing either node 5 with probability 0.4 or node 6 with probability 0.6. If node 5 is chosen, continuing to node 7 and ending at P yields the set of edges $X_1 = \{e_{1,3}, e_{3,4}, e_{4,5}, e_{5,7}, e_{7,P}\}$.

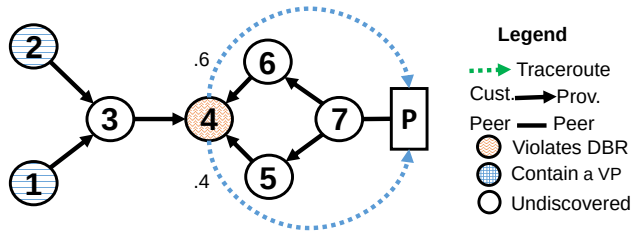


Fig. 5. Traceroutes from vantage points are randomly routed from AS 4 independently among the two outgoing links according to the marked probabilities.

Stochastic Maximum Coverage. The above problem is a special case of the *stochastic maximum coverage* problem. Stoc-Max-Cover$_k$ is a variant of Max-Cover$_k$ where the input is an integer k and $\mathcal{A} = \{A_1, A_2, \ldots\}$ where A_i is a random set chosen according to some known distribution p_i. The goal is pick $\mathcal{A}' \subset \mathcal{A}$ to maximize $\mathbb{E}\left[|\cup_{A \in \mathcal{A}'} A|\right]$.

It can be shown that a natural extension of the greedy algorithm that, at each step, picks the set with the largest expected increase in the coverage, achieves a $1 - 1/e$ approximation [5]. However, note that using this algorithm in our context requires us to be *adaptive*, i.e., we select a vantage point, perform a measurement from that vantage point, and see the result of this measurement before selecting the next vantage point. This is in contrast to a *non-adaptive* algorithm that must choose the full set of k vantage points before running any measurements. There is a trade-off between these two approaches: on the one hand, an adaptive approach may provide a solution of strictly better quality than a nonadaptive algorithm since it has strictly more information.

On the other hand, the adaptive algorithm is slower since after the adaptive algorithm chooses each measurement, it must wait for the traceroute to finish before computing the next measurement. This serial computation and measurement requirement prevents parallelization. Contrast this with the nonadaptive algorithm, which can immediately generate a complete schedule of measurements without actually performing any traceroutes. This schedule can be used to perform the actual traceroutes in any order or even in parallel. This flexibility makes the non-adaptive algorithm desirable in some cases despite its strictly worse solution quality. For these reasons, we implement both versions and evaluate their solution quality theoretically and experimentally. As mentioned above, the adaptive greedy algorithm is guaranteed to achieve an approximation factor of at least $1 - 1/e$. The non-adaptive greedy algorithm is guaranteed an approximation factor of at least $(1 - 1/e)^2$; this follows from work by Asadpour and Nazerzadeh [5] on the more general Stoc-Max-Cover$_k$ problem.

3.6 Evaluating the Greedy Algorithm

We now evaluate the performance of our greedy algorithm in discovering paths towards a given prefix P. The results of the measurements dictated by the algorithm is a directed acyclic graph (DAG) rooted at P. We evaluate the performance of our algorithm and others based on the number of nodes observed in the per-prefix DAGs. We compare our greedy algorithm to common strategies of exploring network topology like random vantage point selection and geo-distributed selection. The geo-distributed strategy selects probes from as many countries as possible *i.e.,* it will not pick a second probe from a given country until all other countries have been selected.

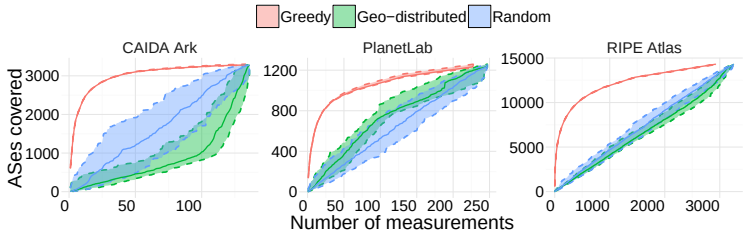


Fig. 6. The number of ASes discovered in the DAGs for different measurement budgets and different measurement strategies, across three measurement platforms: (1) CAIDA Ark (2) PlanetLab, and (3) RIPE Atlas. Results shown for content-heavy prefixes.

We first consider how well the greedy algorithm performs relative to random or geo-distributed measurement strategies. Figure 6 shows the number of ASes discovered in DAGs for destination prefixes for different measurements budgets. The solid line represents the median and dashed lines containing the shaded area show the 5^{th} and 95^{th} percentiles. We perform this evaluation using algorithmic simulation [18] but in Appendix B we show that simulated paths and coverage results match well with actual measurement values.

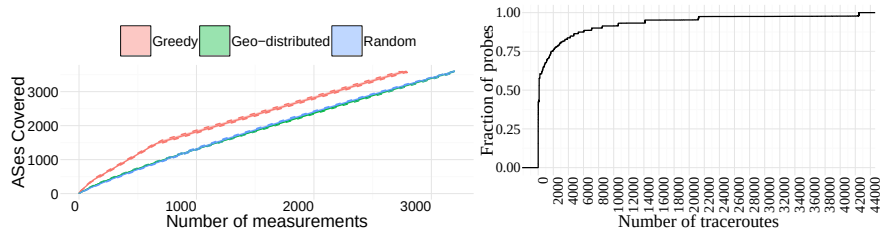
The greedy strategy covers more ASes for a given measurement budget. We find that for the same number of measurements (k = 500), our greedy strategy is able to cover 4 times more ASes than both random and geo-distributed strategies (Figure 6). The random strategy observes a linear increase in the number of ASes covered for a given probing budget, whereas the greedy strategy selects probes that will increase coverage the most early on. After a certain number of measurements, the benefit of adding more measurements shows diminishing returns.

Results are robust across measurement platforms. We evaluate the predicted coverage of the greedy algorithm across different measurement platforms in Figure 6. We find that across the three measurement platforms, the greedy probe selection algorithm maximizes the number of ASes covered for a given measurement budget. Random and geo-distributed perform similarly in all platforms, with the exception of CAIDA Ark, where forcing the system to measure from all countries actually results in lower coverage than would be seen by the random strategy. In Appendix C we show the additional coverage from greedy measurements over the public datasets from various measurement platforms.

Results are robust across prefix types. We find that our results are robust across different destination prefix types (Appendix, Figure 16).

Benefit of including single homed stub ASes. We add single-homed stub customers of ASes containing vantage points to the discovered topology. Since these stubs are single homed, knowing the path from their provider to the destination prefix ensures knowledge of the path from the stub as well. We evaluate the coverage of the greedy algorithm in the absence of gains from single homed ASes. Figure 7(a) shows that our algorithm clearly performs better than other measurement strategies even when excluding coverage gains of single-homed stubs.

Scaling PredictRoute. We now evaluate the load imposed on RIPE Atlas probes if a maximum of 500 (k = 500) PredictRoute proposed optimized measurements are run towards all destination ASes on the Internet. Figure 7(b) shows the distribution of number of traceroutes per probe, if the per-destination AS measurement budget is capped at 500 (we run a maximum 500 traceroutes towards any destination AS). We see that 75% of RIPE Atlas probes would need to execute less than 2000 traceroutes. We hope that this analysis can inform the placement of vantage points by RIPE Atlas in ASes where the burden on existing probes is high.



(a) (b)

Fig. 7. Figure (a) shows ASes covered in the DAGs of prefixes when excluding coverage gains from counting single-homed stub ASes. Figure (b) Distribution of the number of traceroutes that each RIPE probe would need to run if the greedy algorithm was scaled to all destination ASes on the Internet.

4 PATH PREDICTION

We outline how PredictRoute uses empirical data to predict paths. Section 4.1 explains how PredictRoute builds destination specific graphs and auxiliary tables to capture the routing behavior towards a prefix. Section 4.2 details how PredictRoute uses this information to define Markov chains for predicting paths between source and destination hosts on the Internet. Section 4.3 explains how PredictRoute use simulations to fill in gaps in the Markov chains derived from traceroutes. This enables PredictRoute to answer all path prediction queries towards a destination prefix.

4.1 Constructing per-prefix DAGs

We gather a set of traceroutes publicly available on measurement platforms and those run in the topology discovery stage of PredictRoute (§3). We aggregate traceroute hops into BGP routed prefixes (or ASes) to make up the nodes in the DAGs. The process of converting IP paths to prefix-level (or AS-level) paths is involved and requires handling of complex corner cases. We have summarized our procedure in Appendix A.

Let $\mathcal{P}$ be the set of discovered prefix-level paths derived from traceoutes. The process of constructing trusted per-destination graphs has two main components, generating the graph itself and computing the auxiliary *transition tables* that will be used to predict the sequence of edges that a path will take in the graph towards the destination prefix.

(1) **DAG Construction:** Take the union of all edges in the paths in $\mathcal{P}$ to form the directed acyclic graph $\mathcal{D} = (V, E)$.
(2) **Basic Transition Table:** For each edge $e \in E$, let c_e be the count of the number of paths including e.

Temporal transition tables and compression. We are interested in predicting paths both based on the entirety of the observed data and based on data observed during a window of time. To support the latter, we augment our basic transition tables with an additional dimension: time. Let $t \in \{1, 2, \ldots, T\}$ index the relevant time period (*e.g.,* the last month or year) at the required resolution (*e.g.,* hours or days). Let $\mathcal{P}_t$ be the set of discovered paths at time t and let c_e^t be the number of paths in $\cup_{t' \geq t} \mathcal{P}_{t'}$ that includes e. By defining c_e^t in this way, note that $c_e^{t_2} - c_e^{t_1}$ is the number of paths in $\cup_{t_1 \leq t' < t_2} \mathcal{P}_{t'}$ that include e and this can be computed in $O(1)$ time rather than in $O(t_2 - t_1)$ time.

Unfortunately, storing $c_e^1, c_e^2, \ldots, e_e^T$ rather than just c_e increases the space to store the tables by a factor T and this may be significant. To ameliorate this situation, note that c_e^t is monotonically decreasing with t and hence it suffices to only store values for t where $c_e^t \neq c_e^{t-1}$ as the other values can be inferred from this information. By trading-off a small amount of accuracy we can further

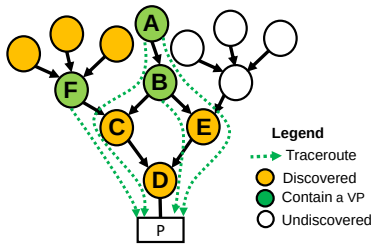


Fig. 8. A DAG constructed from trusted traceroutes. Vantage point A sends two traceroutes which follow paths $ABCD$ and $ABED$. B sends one traceroute with path BED and F sends one traceroute with path FCD.

reduce the space as follows. Suppose we are willing to tolerate a $1+\epsilon$ factor error in the values of c_e^t. Then, round each c_e^t to the nearest power of $1+\epsilon$ and let $\tilde{c}_e^t$ be the resulting value. Then only storing $\tilde{c}_e^t$ for values of t where $\tilde{c}_e^t \neq \tilde{c}_e^{t-1}$ allows every c_e^t to be estimated up to a factor $1+\epsilon$ whilst only storing at most $1+\log_{1+\epsilon}(c_e^1)$ different values.

4.2 Path Prediction via Markov Chains

Learning transition probabilities. The graph for a given prefix P derived from traceroutes towards P defines the structure of the Markov chain used by PredictRoute to model the routing behavior towards the prefix. PredictRoute computes the transition probabilities of each Markov chain using Maximum Likelihood Estimation (MLE), given the traceroute counts stored in edge transition tables. With each edge $e \in E$ we assign a probability $p_e = c_e / \sum_{f \in N_u} c_f$ where N_u is the set of outgoing edges of u. If the user wishes to only train using traceroutes in a time window $\{t_1, t_1+1, \ldots, t_2-1\}$ we set $p_e = (c_e^{t_1} - c_e^{t_2}) / \sum_{f \in N_u} (c_f^{t_1} - c_f^{t_2})$.

Once the Markov models are trained, PredictRoute obtains one Markov chain per destination prefix, where the states of the Markov chain represent BGP prefixes and the end state is the destination prefix P. Edges between the states are evidence of traffic towards P traversing them and the edge transition probabilities define the likelihood of traffic traversing that edge. An example Markov chain is shown in Figure 8. In this chain A, B, C, and D are prefixes. We want to calculate the probability that a packet sent from source prefix A to destination prefix D follows the path $A \rightarrow B \rightarrow C \rightarrow D$.

Defining path probabilities. PredictRoute assumes probabilistic routing obeys a first-order Markov property. In other words, the probability of a packet choosing a next hop towards destination P only depends on the current hop. This dependence on current hops rather than all prior hops is inspired by the prevalence of next-hop routing on the Internet [19]. For the example in Figure 8, this first-order Markov assumption allows us to express the probability of the specified path as

$$\Pr(A, B, C, D) = \Pr(A) \cdot \Pr(B|A) \cdot \Pr(C|B) \cdot \Pr(D|C)$$

Since we have specified that the path begins at A, $\Pr(A) = 1$. In this case, $\Pr(B|A) = 1$ since all traceroutes originating from A go to B. $\Pr(C|B) = 1/3$ since there are 3 traceroutes that go through B, and one of them has next hop C. $\Pr(D|C) = 1$ since both of the traceroutes that go through C have a next hop of D. So $\Pr(A, B, C, D) = 1/3$.

Inferring most likely sequence of states. A naive way of predicting paths from prefix Markov chains would enumerate all paths from the source node S to the destination prefix D and return the one with the highest probability. However, a graph can have an exponential number of paths between the source and the destination, making this approach prohibitively expensive. A more

efficient approach is as follows: for each edge $e = (u, v)$ define the length $\ell_e = -\log(p_e)$. This weight is always non-negative and will be high if $p_e = \Pr(v|u)$ is low and vice versa. Furthermore, the probability of a path is inversely proportion to its length. We then find r shortest paths using Yen's algorithm [38] where the length of e is set to ℓ_e; these correspond to the r paths with the highest probability. We set $r = 5$ in PredictRoute's current implementation. PredictRoute returns a ranked list of these paths and their respective probabilities in response to a path query.

4.3 Splicing Empirical and Simulated Paths

If PredictRoute is faced with a query for the AS-level path between a source AS and a destination prefix such that the source is not a state in the Markov chain of the destination prefix, it cannot predict the path using the chain alone. In such situations we query BGPSim [18] for a policy compliant path from the source AS to the destination prefix. We keep each hop of this path, starting from the source AS, until we reach an AS that is present in the destination's Markov chain. We predict the path between the ASN at the splice point and the destination prefix using methods described in the previous subsection. In this manner, PredictRoute can still return probability-ranked paths for such a query by considering the BGPSim splice of the path as fixed (with transition probabilities of 1).

5 PREDICTROUTE SYSTEM EVALUATION

In this section we discuss the implementation details of PredictRoute, including the construction of per-prefix Markov chains and computing the edge transition probabilities. We then characterize the Markov chains and demonstrate the accuracy of the paths they predict.

5.1 Implementation

Training. For training PredictRoute's per-prefix DAGs, we acquire all public traceroutes available on RIPE Atlas from December 25, 2018 until March 4, 2019. These traceroutes include measurements run by the developers of the platform and user-defined measurements. The scale of these measurements is large, with nearly 2.5-3 million traceroutes being run on the platform every hour. In this manner, we use 4.5 billion public traceroutes to learn the structure of 545,168 Markov chains, one for each routed prefix on the Internet. In Appendix A we describe the procedure for converting traceroutes to prefix-level and AS-level paths. The edges of the Markov chains store the first order transition probability, derived by Maximum Likelihood Estimation on the set of traceroutes.

We build prefix Markov models at two different granularities. In this first, all nodes in the graph, including the destination prefix are BGP routed prefixes. In the second kind, all nodes in the graph except the destination prefix represent ASes. Our goal with these two kinds of graphs is to aggregate information at different levels, allowing the users to pick which works for their application.

In the worst case, a prefix graph can have as many nodes as the number of prefixes on the Internet ($\approx 500,000$). This size is fairly large compared to the AS-level graphs which in the worst case have as many nodes as the ASes on the Internet($\approx 60,000$). However, some applications require fine-grained paths at the prefix-level while others only need AS-level paths (§7).

Characteristics of PredictRoute's destination prefix graphs. As we train the PredictRoute graphs on public traceroutes on RIPE Atlas, we measure the impact of duration of data collection on the size of graphs learned. Figure 9(a) shows the distribution of graph sizes as the duration of data collection increases from 1 week to 8 weeks. Each hour of results adds 3 million traces to PredictRoute's processing pipeline, so we stop at 9 weeks of data for the evaluation of prefix graphs. We note that, in practice prefix graphs have $< 15,000$ nodes, much lesser than the theoretical worst case. The explanation for small graph sizes is two-fold. First, the public traceroutes only provide

(a) Growth of destination graphs sizes with training duration.

(b) Inter-AS cyclical behavior is rare and short-lived.

(c) PredictRoute's accuracy of predicting prefix-level network paths.

Fig. 9. PredictRoute prefix graph characteristics.

partial coverage of the topology and second, Internet paths are shrinking due to the flattening of the Internet [16].

While our training process discards all circuitous traceroutes, interestingly, we observe some of the learned prefix graphs contain cycles. These cycles motivate the inference methodology we use for predicting paths from the graphs (discussed at the end of this section). While several of these cycles were found among nodes belonging to the same ASes, we found a few that crossed AS boundaries. Cyclical structure across AS boundaries suggests routing instability or delayed BGP convergence. We measure the *age* of these cycles as the common duration of time when all edges of the cycles were observed in traceroutes. Figure 9(b) shows the distribution of inter-AS cycle age in hours. We note that these cycles are often short lived and rare. We provide insights into one such inter-AS cycle and think PredictRoute's graphs provide a new lens to study routing instabilities on the Internet (§8).

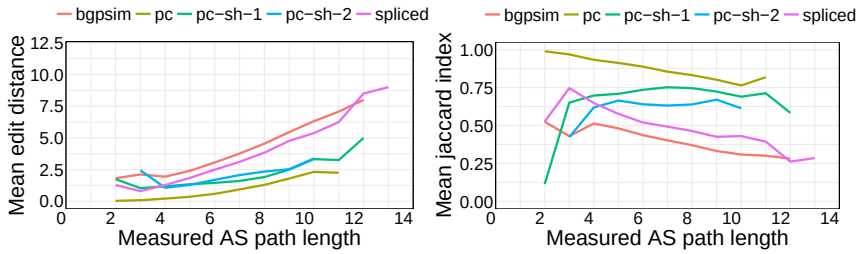
Inference. The problem of predicting the network path from a source node to the destination prefix in the Markov chain, is that of finding the likeliest sequence of states from the source to the destination state. Since enumeration of paths in a potentially cyclical Markov chain can be expensive, we adapt Yen's r-shortest paths algorithm to our problem for finding r most likely paths between the source and destination. The likelihood of a path is calculated with the first order Markov assumption as described in §4.2. Note that Yen's algorithm returns only loop-less paths, ensuring that even if a destination prefix graph contains cycles PredictRoute will never predict cyclical paths.

5.2 Accuracy of Predicted Paths

To evaluate the accuracy of inferred paths from prefix-level PredictRoute graphs, we randomly choose a subset of source and destination prefixes, ensuring we have not measured this path in our training data. We also ensure that the source prefixes chosen have an active RIPE Atlas probe in them. This gives us a set of paths we have not trained on but have the ability of both predicting and evaluating against a measured path. We compare the measured and predicted paths using their edit distance and Jaccard Index (§6).

Prefix-level paths. Figure 9(c) shows the edit distance between the prefix-level predicted path and the measured path for different path lengths. We note, for paths shorter than 10 prefix hops, median edit distance between PredictRoute's predicted path and the measured path is only 1.

AS-level paths. We now evaluate the accuracy of AS-level graphs towards a prefix. Learned in a similar manner as prefix-level DAGs, the only difference is that these DAGs are aggregated at



(a) Edit distance between measured and predicted paths. Lower implies higher accuracy.

(b) Jaccard Index of measured and predicted paths. Higher implies higher accuracy.

Fig. 10. Differences between measured and predicted paths from PredictRoute DAGs (*pc*), simulated paths by BGPSim [18] (*bgpsim*), PredictRoute paths spliced with simulation (*spliced*) and those predicted by appending 1 (*pc-sh-1*) or 2 (*pc-sh2*) single homed stubs.

AS-level hops. PredictRoute's AS path predictions differ from the measured path by at most 1 hop, 75% of the time (Figure 10(a)).

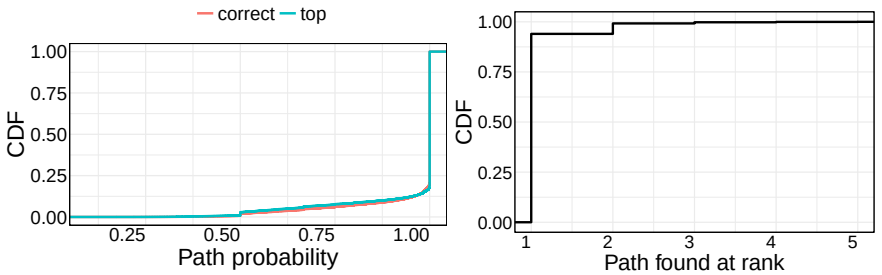
Incorporating simulation data. Simulation-based path prediction can answer all path prediction queries at the AS-level but offers low prediction accuracy. A key goal of PredictRoute is to answer queries for arbitrary paths on the Internet more accurately than state-of-the-art simulation approaches [18]. However, PredictRoute's destination-specific models are trained using traceroutes that are limited by the measurement budget and availability of vantage points. Thus, some path prediction queries may not be answered from the destination-specific routing models alone. We consider two methods of incorporating simulated paths into PredictRoute. First, we consider returning the entire simulated path towards the destination prefix when there is no path from the requested source AS in the DAG. Second, we return the simulated path, up until we observe an AS in the DAG. Beyond this AS, we return the path as predicted by the DAG. The idea being that with destination routing this allows us to base the latter half of the path on empirical data. We note that splicing with simulated paths is only possible when answering AS-level path queries.

We evaluate our approach for incorporating simulated path data using ASes that are not in the DAG for a given prefix, but that contain a RIPE Atlas probe. This results in a set of 2,231 source destination pairs where we can measure the path and then compare with the predicted path.

Figure 10(b) shows the edit distance between measured paths and those returned by the BGP simulator (*bgpsim*) or the path produced by splicing the simulated path with the path found in the DAG (*spliced*). We observe that the PredictRoute spliced paths get at most two AS hops wrong over 82% of the time whereas simulated paths get at most two hops wrong only 50% of the time. This shows illustrates the benefits of using an approach that combines measurement and simulations even when complete measurement does not cover all paths.

Relationship between accuracy and path probability. We analyze the probability of the highest ranked path in PredictRoute. This indicates the confidence value associated with PredictRoute's path prediction. Figure 11(a) shows the weight of the highest ranked path for paths where we predicted correctly and those where we did not. We see that the predicted path has a confidence of 1.0 in for nearly 80% of the paths.

We next consider how often expanding the number of paths returned increases our accuracy. Figure 11(b) shows the rank at which we find the correct path in PredictRoute's predictions. Over 90% of accurate paths are found at rank 1 highlighting that the most likely paths in PredictRoute's predictions are accurate in majority of the cases.



(a) Confidence of predicted paths. Over 80% of the paths have a confidence value of 1.0.

(b) 90% of accurate paths predicted by PredictRoute were the most likely (ranked 1).

Fig. 11. (a) the confidence value of the top ranked path and (b) the rank at which the correct path is discovered.

6 COMPARISON WITH RELATED WORK

In this section, we compare the performance of PredictRoute with two canonical works in this area, iPlane [26] and Sibyl [24]. We evaluate the three systems based on their accuracy *i.e.,* how close is the returned path to the actual path) and their coverage (*i.e.,* for what fraction of queries can the system answer).

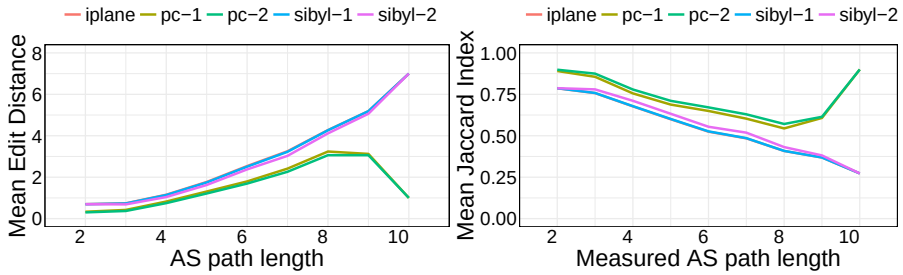
6.1 Comparison Methodology

Similar to PredictRoute, iPlane consumes traceroutes and related metadata to build an *atlas* of paths. Using the atlas, iPlane predicts PoP-level paths by splicing paths in the atlas. A key piece of data consumed by iPlane is the IP-to-PoP level mapping, built from traceroutes and active measurements (*e.g.,* DNS queries) towards intermediate traceroute hops. iPlane's spliced paths serve as inputs for Sibyl [24] which then augments these predictions with confidence scores using a supervised learning algorithm [31].

To compare the accuracy of PredictRoute's path prediction with iPlane, we train PredictRoute's DAGs and iPlane's atlas on the same set of traceroutes. We extract features from iPlane's predictions to train Sibyl's supervised learning model and use the predictions of the model (*i.e.,* the confidence scores) to evaluate Sibyl's prediction accuracy using paths with highest confidence values.

When comparing to Sibyl and iPlane, we faced the challenge that the IP-to-PoP implementation required by these systems was not available to run on present-day traceroutes. Thus, to perform the evaluation, we trained PredictRoute on set of $\approx$ 20 million traceroutes which were used in 2016 to evaluate Sibyl. The authors gave us access to the IP-to-PoP mappings for this training data, allowing us to compare both PredictRoute and the related work on the same training dataset. These traceroutes were performed from a wide range of vantage points: PlanetLab, RIPE Atlas and traceroute servers and looking glasses. To isolate the effectiveness of path prediction in these systems, we use the same set of 20 million traceroutes to train and test each system.

We performed five-fold cross-validation by randomly dividing the set of traceroutes into 5 groups (each with 20% of overall traces) and training the path prediction systems (iPlane, Sibyl and PredictRoute) on 4 of the 5 groups (80% of the data). We evaluate the performance of the systems on the remaining 20% of paths. We repeat this process such that each of the five folds is used for testing once. In each round, the systems were trained on 16 million and tested on 4 million traceroutes. Further details about the evaluation and its limitations are covered in Appendix D.



(a) Edit distance between measured and predicted paths (lower implies higher accuracy).

(b) Jaccard Index of measured and predicted paths (higher implies high accuracy).

Fig. 12. Comparison of accuracy of PredictRoute, iPlane, and Sibyl. We find that Sibyl's weighting of returned paths improves accuracy over iPlane, but PredictRoute provides paths that are closer to those observed in measurements.

6.2 Comparing accuracy of path prediction

Using the 5-fold cross validation methodology, we compare the accuracy of iPlane, Sibyl, and PredictRoute. For a given traceroute in the test data, we query each of the 3 systems and compare the predicted paths with the measured path in the test set. Since PredictRoute gives a list of path predictions ranked by their likelihood, we compare its accuracy on the highest ranked path *pc-1* and its accuracy when the second highest ranked path is included as an option *pc-2*. Similarly, Sibyl provides confidence values ($0 < c < 1$, predicted by RuleFit) for predicted paths. We consider accuracy of the top 2 paths (*sibyl-1*, *sibyl-2*) predicted by Sibyl.

We evaluate the systems' accuracy based on two distance metrics and compare these metrics relative to the length of the path being predicted.

1. Levenshtein distance (edit distance): This metric measures the minimum number of single-element edits (insertions, deletions or substitutions) required to change one sequence of ASes into the other. This metric takes into account the ordering of ASes on the paths being compared. In Figure 12(a), we see that PredictRoute performs better than iPlane and Sibyl for all path lengths in terms of mean edit distance. We note that as the gap in the accuracy between PredictRoute and other systems increases as the AS path length increases. However, we note that the number of paths with a length of ≥ 5 is small, with only 10% of 2 million paths having length ≥ 5 ASes.

2. Jaccard Index: This metric evaluates closeness to the measured path if order of ASes on the path is not important (*i.e.,* the predicted path contains the ASes in the measured path). Figure 12(b) shows the distribution of Jaccard indices of predicted and measured paths for PredictRoute, Sibyl and iPlane. Similar to edit distance, we observe PredictRoute performs better than iPlane or Sibyl, in terms of Jaccard similarity of the predicted and measured paths.

We attribute PredictRoute's increased accuracy to the way that iPlane combines network paths in its path prediction. To predict a path towards prefix P, the initial path segment used by iPlane (from source node to intermediate hop) can be derived from any traceroute, not necessarily one that is destined to prefix P. Since routing is largely destination-based, the first splice of the path may not match the route the source would take to P. PredictRoute avoids this issue by predicting paths using the DAG for P which is derived from traces destined towards P only.

6.3 Comparing coverage of path queries

We evaluate the fraction of path queries that PredictRoute and iPlane are able to respond to, given the same training dataset. We find that in each fold of the cross-validation, we query the systems for ≈2 million paths that they have not been trained on. iPlane provides paths for 70% of the queries. PredictRoute responded to 100% of the path queries. The reason that PredictRoute's is always able to predict a path is its ability to use simulated paths to fill in path segments that have not been directly observed. We discuss details of how we incorporate simulated path segments into our path predictions in §5.2. Since Sibyl leverages iPlane's predictions, it has the same coverage as iPlane.

7 CASE STUDIES

In this section we demonstrate the impact of PredictRoute on real-world applications.

PredictRoute for Refraction Networking (RN). Refraction networking [37] is a recent technique to circumvent Internet censorship that incorporates the circumvention infrastructure into routers on the Internet. This technique is considered more resilient to blocking by censors since it is hard to block individual routers on the Internet, while blocking the sources and destinations of packets is relatively easy.

A key problem faced by Refraction Routing deployments today [14] is to place refraction routers in large ISPs such that client traffic gets intercepted by them. If client traffic follows a path without the refraction router on it, it leads to the failure of the refraction routing session.

We worked with the largest ISP-scale deployment of refraction routing [14] to use PredictRoute for predicting if a client refraction routing session will be successful. To predict a successful refraction routing connection, we use PredictRoute to predict the path between the client and refraction router. If the PredictRoute predicted path crosses specific prefixes within the deploying ISP, we conclude that the connection went via a refraction router and was successful (except for other non-networking failures). We find that in the current deployment of clients, PredictRoute can predict the prefix hop inside the deploying ISP which client traffic took when a RN session was successful, 100% of the time. In future, we are working towards incorporating the PredictRoute API into RN software for improved client performance.

PredictRoute to defend against routing adversaries. Researchers have found that anonymous communication via Tor [11] is susceptible to network-level adversaries launching routing attacks [34]. Several defenses against such attacks have been proposed that aim to avoid Tier-1 providers [6], use simulated BGP paths to avoid network-level adversaries [28, 33], etc. PredictRoute's AS-level path prediction is highly accurate and readily available as a REST API which can be incorporated into Tor client software for defending against network-level adversaries.

8 DISCUSSION AND FUTURE WORK

In addition to their utility for path prediction, PredictRoute prefix graphs capture routing behavior in a novel way. We believe they can be used to revisit several classical problems in inter-domain routing. For instance:

BGP atoms. The networking research community has long studied the right granularity for modeling routing behavior on the Internet. One proposal is to find a set of routers that route towards the Internet similarly, called BGP atoms [3]. We note that since PredictRoute has a view of the routing behavior of all prefixes on the Internet, using measures of graph similarity across prefixes, PredictRoute's Markov chains can potentially provide a way to infer BGP atoms.

Analyzing routing convergence. In Section 5.1 we described the existence of cycles in PredictRoute's per-destination graphs. While these cycles are rare across ASes and have very short duration, we think they offer a new perspective on the analysis of BGP route convergence. Figure

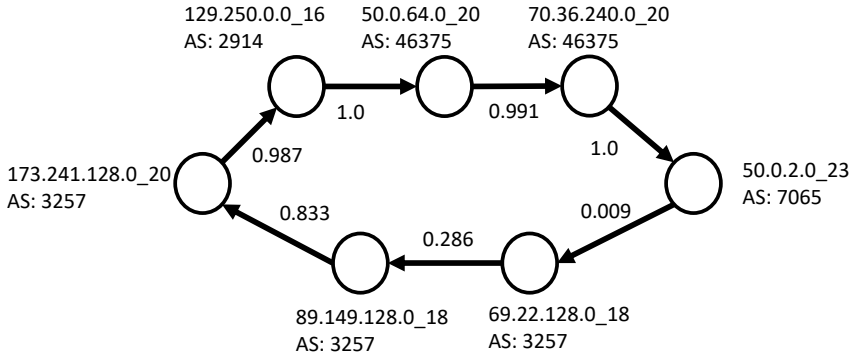


Fig. 13. A cycle observed in the PredictRoute routing model of prefix 122.10.0.0/19. This cycle is across 4 ASes and lasted for 3 hours, as measured by traceroutes. Node ASNs, prefixes and edge probabilities are annotated.

13 shows one such cycle observed in the PredictRoute graph for prefix 122.10.0.0/19 which lasted for 3 hours. Combining PredictRoute's data plane analysis with BGP announcements can help us identify the cause of these cycles and how long it took for them to be resolved in the control as well as the data plane.

9 BACKGROUND

In this section we discuss prior work that addresses the challenge of predicting network paths via simulations and measurement-based path prediction.

9.1 Simulations for path prediction

A number of studies rely on modeling and simulation of inter-domain paths. These studies model the AS-level Internet topology as a graph $G = (V, E)$ where the vertices represent ASes and the edges represent connections between them. The edges in the graph are annotated with business relationships between the ASes (*e.g.,* customer-provider, peer-peer). Using this graph abstraction and a model of routing policies like the Gao-Rexford model [15], researchers have simulated the paths on empirical network topologies, *e.g.,* the CAIDA topology [8]. Simulations of network paths face scaling challenges due to the size of the network graph ($|V| \approx 50K$, $|E| \approx 140K$) and the fact that routing on the Internet does not follow shortest paths. Approaches to scale simulations include custom simulators [22, 35], methods to scale down the topology [25], and specialized algorithms to enable efficient computation of policy compliant paths on the entire Internet graph [18].

The accuracy and limitations of simulated network paths have been investigated by a study which finds that 65% of measured paths match those returned by simulators [4]. However, in cases where measurement-based path predictions are not possible due to the lack of appropriate vantage points, simulations can provide a coarse-grained approximation of the actual network path. In fact, PredictRoute leverages an algorithmic network path simulator, BGPSim [18] as the fall-back for path prediction when a more accurate path prediction is not possible.

9.2 Measurement-based path prediction

iPlane [26] predicts the path from source S to destination D by combining paths from S to an arbitrary destination with a path from an arbitrary source to D. This technique, referred to as *path splicing* is used to provide predictions of point-of-presence or PoP-level paths. We note that the first section of the spliced path may be inaccurate since it is derived from a measurement originating at S towards an *arbitrary destination*, as opposed to the desired destination D.

In subsequent years, researchers have built iPlane Nano [27] to reduce the memory footprint of iPlane, enabling the use of such a system on personal computers and mobiles. Unlike iPlane which stored an atlas of paths, iNano stores a graph of connectivity information (links between PoPs, ASes) observed from in traceroutes. To predict paths, iNano performs a shortest path search on the graph and predicts only those AS-level paths that conform to BGP policies and known constraints on paths *e.g.,* valley free routing. More recently, researchers leveraged iPlane's path prediction to build Sibyl [24] for optimally discovering measurements that satisfy complex path queries. In this work, the authors noted the low accuracy of path splicing formulate iPlane Nano (68%) and use supervised machine learning to help improve accuracy when multiple spliced paths are available.

9.3 Measurement platforms and data sets

The rise of inexpensive, small form factor computing devices (*e.g.,* Raspberry Pi [29]) has lowered the bar for deploying vantage points in a range of networks. Prime examples are RIPE Atlas [30] and Ark [7], which have vantage points in 3,200 and 142 ASes, respectively. While the number of vantage points and locations has increased with these platforms, their constrained hardware[1] necessitates judicious use of their resources. Indeed, a recent study highlights the impact of load on measurements run in parallel on RIPE Atlas nodes [21].

To facilitate more efficient use, many of these platforms make their measurements publicly available. For example, RIPE Atlas publishes traceroutes to destinations spread across nearly 500 ASes from vantage points in over 3,200 ASes. RIPE Atlas also implements a credit system to limit the load individual users can place on the platform. In this study, we develop techniques to make more efficient use of our measurement budget on the RIPE Atlas platform. We also re-use the public measurements on RIPE Atlas for building prefix routing models [32].

10 CONCLUSION

This paper introduces PredictRoute, a system that predicts network paths between arbitrary hosts on the Internet. We show analytically that it makes efficient use of available measurement platforms to maximize measurement inputs to its path prediction system. We empirically verify that PredictRoute's path predictions improve upon the state of the art, both in terms of path query coverage and in accuracy of predicted paths. We also demonstrate PredictRoute's utility in improving applications for circumventing Internet censorship and revisiting classical inter-domain routing problems. We have released PredictRoute's code and prototype for use in Internet applications and other studies.

11 ACKNOWLEDGEMENTS

The authors thank Priya Mahadevan, Rishab Nithyanand, the anonymous reviewers and our shepherd John C. S. Lui for their feedback on this work. We thank Italo Cunha for his help in performing comparisons between PredictRoute and related work. This research was supported by NSF grants CCF-1934846, CCF-1908849, CCF-1637536, CCF-1725543, CSR-1763680, CCF-1716252, CNS-1938709 and a Google faculty research award. Rachee Singh was supported by the Google PhD fellowship.

REFERENCES

[1] CAIDA's Prefix to ASN dataset. https://www.caida.org/data/routing/routeviews-prefix2as.xml.
[2] Peeringdb. https://www.peeringdb.com/.

[1] RIPE's v1 and v2 probes have 167MHz CPU, 8MB or 16MB of RAM. The new and more powerful v3 probes have 400MHz CPU, 32MB of RAM.

[3] Y. Afek, O. Ben-Shalom, and A. Bremler-Barr. On the structure and application of bgp policy atoms. In *Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment*, IMW '02, pages 209–214, New York, NY, USA, 2002. ACM.
[4] R. Anwar, H. Niaz, D. Choffnes, I. Cunha, P. Gill, and E. Katz-Bassett. Investigating interdomain routing policies in the wild. In *ACM IMC*, 2015.
[5] A. Asadpour and H. Nazerzadeh. Maximizing stochastic monotone submodular functions. *Management Science*, 62(8):2374–2391, 2016.
[6] A. Barton and M. Wright. Denasa: Destination-naive as-awareness in anonymous communications. *Proceedings on Privacy Enhancing Technologies*, 2016(4):356–372, 2016.
[7] CAIDA Ark. http://www.caida.org/projects/ark/.
[8] CAIDA AS Relationship dataset. http://data.caida.org/datasets/as-relationships/.
[9] K. Chen, D. R. Choffnes, R. Potharaju, Y. Chen, F. E. Bustamante, D. Pei, and Y. Zhao. Where the sidewalk ends: Extending the internet as graph using traceroutes from p2p users. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, pages 217–228, New York, NY, USA, 2009. ACM.
[10] Y. Chiu, B. Schlinker, A. Radhakrishnan, and E. Katz-Bassett. Are we one hop away from a better Iinternet? In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, pages 71–77. ACM, 2015.
[11] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.
[12] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, July 1998.
[13] T. Flach, E. Katz-Bassett, and R. Govindan. Quantifying violations of destination-based forwarding on the internet. In *IMC '12*, 2012.
[14] S. Frolov, F. Douglas, W. Scott, A. McDonald, B. VanderSloot, R. Hynes, A. Kruger, M. Kallitsis, D. G. Robinson, S. Schultze, et al. An isp-scale deployment of tapdance. In *7th {USENIX} Workshop on Free and Open Communications on the Internet ({FOCI} 17)*, 2017.
[15] L. Gao and J. Rexford. Stable Internet routing without global coordination. 2001.
[16] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. The flattening internet topology: Natural evolution, unsightly barnacles or contrived collapse? In *International Conference on Passive and Active Network Measurement*, pages 1–10. Springer, 2008.
[17] P. Gill, M. Schapira, and S. Goldberg. Let the market drive deployment: A strategy for transitioning to bgp security. *SIGCOMM Comput. Commun. Rev.*, 41(4):14–25, Aug. 2011.
[18] P. Gill, M. Schapira, and S. Goldberg. Modeling on quicksand: Dealing with the scarcity of ground truth in interdomain routing data. *ACM SIGCOMM Computer Communication Review*, 42(1):40–46, 2012.
[19] P. Gill, M. Schapira, and S. Goldberg. A survey of interdomain routing policies. *In ACM Computer Communications Review (CCR).*, Jan 2014.
[20] V. Giotsas, M. Luckie, B. Huffaker, and kc claffy. Inferring complex AS relationships. In *ACM IMC*, 2014.
[21] T. Holterbach, C. Pelsser, R. Bush, and L. Vanbever. Quantifying interference between measurements on the RIPE Atlas platform. In *ACM IMC*, 2015.
[22] J. Karlin, S. Forrest, and J. Rexford. Autonomous security for autonomous systems. *Computer Networks*, oct 2008.
[23] E. Katz-Bassett, H. Madhyastha, V. Adhikari, C. Scott, J. Sherry, P. van Wesep, A. Krishnamurthy, and T. Anderson. Reverse traceroute. In *USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, 2010.
[24] E. Katz-Bassett, P. Marchetta, M. Calder, Y.-C. Chiu, I. Cunha, H. Madhyastha, and V. Giotsas. Sibyl: A practical internet route oracle. In *USENIX NSDI*, 2016.
[25] V. Krishnamurthy, M. Faloutsos, M. Chrobak, L. Lao, J.-H. Cui, and A. G. Percus. Sampling large Internet topologies for simulation purposes. 2007.
[26] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An Information Plane for Distributed Services. In *Proc. of Operatings System Design and Implementation*, 2006.
[27] H. V. Madhyastha, E. Katz-Bassett, T. E. Anderson, A. Krishnamurthy, and A. Venkataramani. iplane nano: Path prediction for peer-to-peer applications. In *NSDI*, volume 9, pages 137–152, 2009.
[28] R. Nithyanand, R. Singh, S. Cho, and P. Gill. Holding all the ases: Identifying and circumventing the pitfalls of as-aware tor client design. *CoRR*, abs/1605.03596, 2016.
[29] Raspberry Pi. https://www.raspberrypi.org/.
[30] RIPE Atlas. https://atlas.ripe.net/.
[31] Rule Based Learning Ensembles. https://statweb.stanford.edu/~jhf/R_RuleFit.html.
[32] R. Singh and P. Gill. Pathcache: A path prediction toolkit. In *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, page 569–570, New York, NY, USA, 2016. Association for Computing Machinery.
[33] O. Starov, R. Nithyanand, A. Zair, P. Gill, and M. Schapira. Measuring and mitigating as-level adversaries against tor. In *NDSS*, 2016.
[34] Y. Sun, A. Edmundson, L. Vanbever, O. Li, J. Rexford, M. Chiang, and P. Mittal. Raptor: Routing attacks on privacy in tor. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 271–286, Washington, D.C., Aug. 2015. USENIX

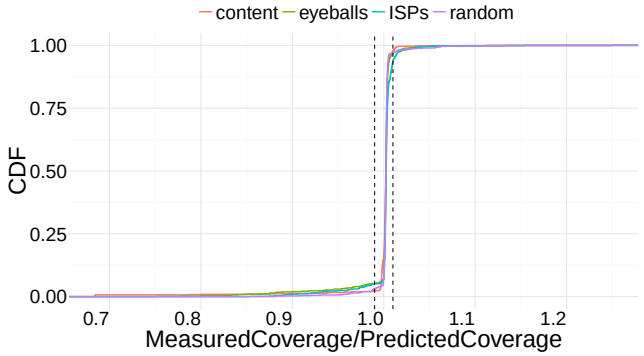


Fig. 14. The measured coverage of AS graphs obtained via running traceroutes is very close to the coverage predicted by BGP policy simulations. Therefore the approximate gain of a measurement used by the greedy algorithm is close the gain obtained by running the measurement.

Association.
[35] M. Wojciechowski. Border gateway protocol modeling and simulation. master's thesis. 2008.
[36] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 30–30, Berkeley, CA, USA, 2011. USENIX Association.
[37] E. Wustrow, S. Wolchok, I. Goldberg, and J. A. Halderman. Telex: Anticensorship in the network infrastructure. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 30–30, Berkeley, CA, USA, 2011. USENIX Association.
[38] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

A EXTRACTING CONNECTIVITY INFORMATION FROM TRACEROUTES

Using traceroutes to extract information about connectivity between prefixes and ASes on the Internet is a hard problem due to several practical issues. In this section we highlight the assumptions we made in processing traceroutes.

- **IP-to-Prefix and IP-to-ASN** We use the CAIDA prefix to ASN dataset [1] to map IP addresses to the longest matching prefix in the BGP routing table. Similarly, we associate prefixes with ASNs that announced them.
- **Dealing with prefix migration** Some prefixes are announced by different ASes at different points of time for several reasons. In order to keep track of the ASN announcing a prefix at time t, we use search the BGP announcements of that prefix closest to t.
- **Dealing with discontinuous traceroutes** Often traceroutes are discontinuous because some networks do not respond to ICMP packets. In our measurements, MegaFon (ASN 31133), Hurricane Electric (ASN 6939) and Level 3(ASN 3356) are the top 3 ASes to not respond to traceroute packets. However, we include discontinuous traces in the PredictRoute graphs in the hope that at a later point, new traces would connect the dangling traceroute to the rest of the graph.
- **Dealing with traceroutes with loops** We only added loop-free traceroutes to the PredictRoute graphs to ensure high accuracy of the system. As a result, the cycles formed in the graphs are a result of 2 or more traceroutes.
- **Dealing with IXPs on path** IXPs are an important part of the inter-domain routing ecosystem. We identify IXP prefixes using aggregated datasets from PeeringDB, PCH etc. PredictRoute uses this information to inform the user if their traffic crossed an IXP.

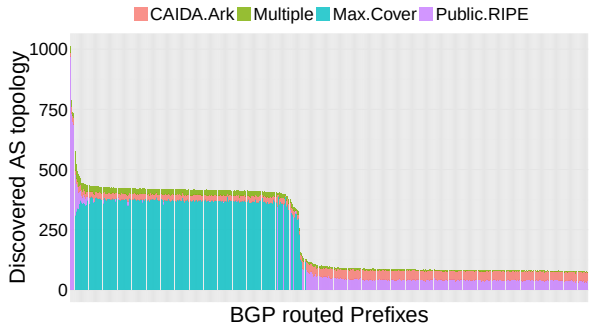


Fig. 15. Number of ASes in the empirically derived directed acyclic graphs (DAGs) for each prefix. Prefixes are ranked by the number of AS included in the DAG and prefixes with less than 75 ASes in their graph are not shown, they are qualitatively similar to the ASes on the right-hand side of the graph.

B COVERAGE PREDICTED BY SIMULATIONS IS VERY CLOSE TO WHAT IS SEEN IN PRACTICE.

A key question we face in using a simulator to compute coverage and select measurements to run, is whether the simulator allows us to make predictions that are close to reality. Figure 14 shows the ratio of the measured and predicted (via simulation) coverage values (*i.e.,* number of ASes covered by the DAGs). We find that in general, the simulated coverage is very close to what we observe in practice. In fact, 92.5% of the time, the predicted coverage is within 1% of the measured values.

C BENEFITS OF AGGREGATING DATASETS

PredictRoute's AS-level DAGs include ASes observed on paths towards the destination prefix. These paths may come from existing traceroute data or measurements we issue ourselves according to the greedy algorithm (§3). Figure 15 shows the size of the DAGs for prefixes with at least 75 ASes in the DAG.

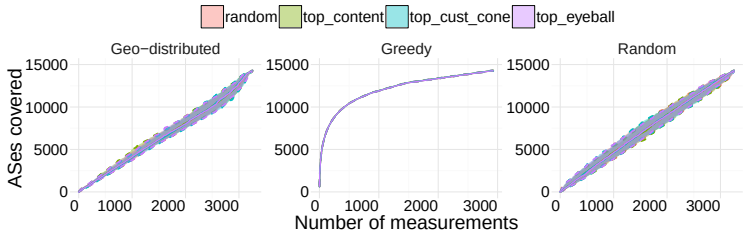


Fig. 16. The number of ASes covered by the DAGs for different measurement budgets and different measurement strategies: (1) Geo-distributed (2) Greedy, and (3) Random and four destination prefix types. The prefix types are: randomly chosen routed prefixes (*random*), prefixes hosting content for Alexa top 100 websites (*top_content*), prefixes announced by ASes with the largest customer cones on the Internet (*top_cust_cone*) and prefixes announced by top eyeball networks (*top_eyeball*). We note that the results are consistent across prefix types.

Breadth of coverage from CAIDA Ark and RIPE Atlas Anchors. We find that CAIDA's Ark platform is able to give us broad coverage of many destination prefixes. This follows from the

fact that this platform is used to traceroute to all globally routed prefixes from a set of 25 ASes containing probes. Similarly, the public RIPE data contains broad traceroute campaigns from 200 anchor probes (higher capacity nodes that can perform more intensive measurements).

In depth measurements of few destinations from RIPE Atlas. The bulk of the RIPE traceroutes we ingest center on a small number of destination prefixes (*i.e.,* campaigns that traceroute from a large number of probes to a single prefix). This follows from the usage model of RIPE Atlas, which facilitates selecting many sources but not necessarily a large pool of measurement targets. Indeed, the largest DAGs we observe are for a prefix 194.0.1.0/24 announced by ASN 42909 (Community DNS) which contains 1015 ASes.

Efficient coverage via our greedy algorithm. Figure 15 also shows the results of running our greedy algorithm for selecting vantage points. DAGs for prefixes where we performed greedy measurements contain an average of 410 ASes (and 6886 single homed stub ASes on an average). This is in contrast to an average of 45 ASes for prefixes where we rely on existing traceroute data. We also note that this gain in terms of ASes is accomplished with a relatively small number of measurements.

D EVALUATION DETAILS

Sibyl adds a layer of ensemble based supervised machine learning on top of iPlane's predicted paths. We used the same implementation library (RuleFit [31]) as was used by the authors of Sibyl. This library is an R-wrapper around a Fortran based implementation. As a first step, we extracted features from path predictions from each fold of cross-validation. We used the same set of features used by the authors of Sibyl [24]. Since the training data in each fold, consists of 16 million traceroutes, the set of data/features used by RuleFit is correspondingly large. Nevertheless we attempted to learn a model from the training data using the same RuleFit implementation as used by Sibyl. This evaluation was done on a dedicated 252GB, 56 core server. However, the Fortran library throws memory corruption errors (SIGSEGV), when RuleFit is trained using the full 16 million traceroutes. We verified that this was not caused by our input data but was due to the large size of the input by reducing the size of the training/test datasets (using random sampling). A smaller training and test data combination leads to successful learning of the model. To resolve this issue, we randomly sampled training and test points in each fold of the evaluation, until we could get RuleFit to learn a model. The size of training data which was a threshold for this was $\approx$ 500,000 training examples. The test data had to be similarly smaller so we again randomly sampled from the 20% test queries and used the learned model to predict the confidence values of those. In this manner, we evaluate Sibyl over $\approx$ 50,000 paths overall. We reduce the testing set as well as the training set in an effort to be fair in our evaluation.

Received February 2021; revised April 2021; accepted April 2021